# Package 'forestFloor'

May 30, 2017

**Type** Package

**Title** Visualizes Random Forests with Feature Contributions

**Version** 1.11.1

**Date** 2017-05-30

**Author** Soeren Havelund Welling

**Maintainer** Soeren Havelund Welling <SORHAWELL@GMAIL.COM>

**Depends**

**Suggests** caret, e1071

**Description** Form visualizations of high dimensional mapping structures of random forests and feature contributions.

**SystemRequirements** OpenGL, GLU Library, zlib

**License** GPL-2

**URL** http://forestFloor.dk

**Imports** Rcpp (>= 0.11.3), rgl, kknn, randomForest

**LinkingTo** Rcpp

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2017-05-30 12:17:27 UTC

## R topics documented:

---

`forestFloor-package` *forestFloor: visualize the random forest model structure*

---

## Description

forestFloor visualizes randomForests models(RF). Package enables users to understand a non-linear, regression problem or a binary classification problem through RF. Any model can be separated into a series of main effect and interactions with the concept of feature contributions.

## Details

| | |
|---|---|
| Package: | forestFloor |
| Type: | Package |
| Version: | 1.9 |
| Date: | 2015-12-25 |
| License: | GPL-2 |

## Author(s)

Soren Havelund Welling

## References

Interpretation of QSAR Models Based on Random Forest Methods, http://dx.doi.org/10.1002/minf.201000173
Interpreting random forest classification models using a feature contribution method, http://arxiv.org/abs/1312.1121

---

`append.overwrite.alists`
*Combine two argument lists*

---

## Description

First argument list is master, second list slave

## Usage

```
append.overwrite.alists(masterArgs,slaveArgs)
```

## Arguments

| | |
|---|---|
| masterArgs | List of arguments, of which will stay unchanged |
| slaveArgs | List of arguments, conflicts with masterArgs will be deleted. Additional args will be appended. |
| | s |

## Details

This function combines to lists of arguments. Conflicts will be resolved by masterArgs.

## Value

List of arguments, being masterArgs appended by slaveArgs

## Author(s)

Soren Havelund Welling

## Examples

```
arglist1 = alist(monkey="happy",telephone.no=53)
arglist2 = alist(monkey="sad",house.no=12)

#this should yield a alist(monkey="happy", telephone.no=53, house.no=12)
forestFloor:::append.overwrite.alists(arglist1,arglist2)
```

---

# as.numeric.factor *Convert a factor to numeric.vector.*

---

## Description

Internal function which will drop unused levels and convert remaining to a number from 1 to n.levels.

## Usage

```
as.numeric.factor(x,drop.levels=TRUE)
```

### Arguments

| | |
|---|---|
| x | Normally a factor, can be a numeric vector(will be output unchanged) |
| drop.levels | Boolean, should unused levels be dropped? |

### Details

Simple internal function, used to direct categorical variables to a 1 dimensional scale.

### Value

A vector of same length, where each category/level is replaced with number from 1 to n

### Author(s)

Soren Havelund Welling

### Examples

```
as.numeric.factor = forestFloor:::as.numeric.factor #import to environment
some.factor = factor(c("dog","cat","monkey")[c(1,3,2,1,3,2,1,1)]) #make factor
a.numeric.vector = as.numeric.factor(some.factor) #convert factor representation.
```

---

## box.outliers *Box Outliers*

---

### Description

Squeeze all outliers onto standard.dev-limits and/or normalize to [0;1] scale

### Usage

```
box.outliers(x, limit = 1.5, normalize = TRUE)
```

### Arguments

| | |
|---|---|
| x | numeric vector, matrix, array, data.frame |
| limit | limit(SD,standard deviation) any number deviating more than limit from mean is an outlier |
| normalize | TRUE/FALSE should output range be normalized to [0;1]? |

### Details

Can be used to squeeze high dimensional data into a box, hence the name box.outliers. Box.outliers is used internally in forestFloor-package to compute colour gradients without assigning unique colours to few outliers. It's a box because the borders uni-variate/non-interacting.

## Value

matrix(n x p) of normalized values

## Author(s)

Soren Havelund Welling, 2014

## See Also

scale()

## Examples

```
box.outliers = function (x, limit = 1.5)  {
  x = scale(x)
  x[ x > limit] =  limit
  x[-x > limit] = -limit
  x = x - min(x)
  x = x/(limit * 2)
  return(x)
}
n=1000 #some observations
p = 5  #some dimensions
X = data.frame(replicate(p,rnorm(n))) # a dataset
Xboxed =box.outliers(X,limit=1.5) #applying normalization
plot(Xboxed[,1],Xboxed[,2],col="#00000088") #plot output for first two dimensions
```

---

# convolute_ff *Cross-validated main effects interpretation for all feature contributions.*

---

## Description

convolute_ff estimates feature contributions of each feature separately as a function of the corresponding variable/feature. The estimator is a k-nearest neighbor function with Gaussian distance weighting and LOO cross-validation see train.kknn.

## Usage

```
convolute_ff(ff,
             these.vars=NULL,
             k.fun=function() round(sqrt(n.obs)/2),
             userArgs.kknn = alist(kernel="epanechnikov"))
```

## Arguments

| | |
|---|---|
| ff | forestFloor object "forestFloor_regression" or "forestFloor_multiClass" consisting of at least ff$X and ff$FCmatrix with two matrices of equal size |
| these.vars | vector of col.indices to ff$X. Convolution can be limited to these.vars |
| k.fun | function to define k-neighbors to consider. n.obs is a constant as number of observations in ff$X. Hereby k neighbors is defined as a function k.fun of n.obs. To set k to a constant use e.g. k.fun = function() 10. k can also be overridden with userArgs.kknn = alist(kernel="Gaussian",kmax=10). |
| userArgs.kknn | argument list to pass to train.kknn function for each convolution. See (link) kknn.args. Conflicting arguments to this list will be overridden e.g. k.fun. |

## Details

convolute_ff uses train.kknn from kknn package to estimate feature contributions by their corresponding variables. The output inside a ff$FCfit will have same dimensions as ff$FCmatrix and the values will match quite well if the learned model structure is relative smooth and main effects are dominant. This function is e.g. used to estimate fitted lines in plot.forestFloor function "plot(ff,...)". LOO cross validation is used to quantify how much of feature contribution variation can be explained as a main effect.

## Value

ff$FCfit a matrix of predicted feature contributions has same dimension as ff$FCmatrix. The output is appended to the input "forestFloor" object as $FCfit.

## Author(s)

Soren Havelund Welling

## Examples

```
## Not run:
library(forestFloor)
library(randomForest)

#simulate data
obs=1000
vars = 6
X = data.frame(replicate(vars,rnorm(obs)))
Y = with(X, X1^2 + 2*sin(X2*pi) + 8 * X3 * X4)
Yerror = 5 * rnorm(obs)
cor(Y,Y+Yerror)^2
Y= Y+Yerror

#grow a forest, remeber to include inbag
rfo=randomForest(X,Y,keep.inbag=TRUE)

ff = forestFloor(rfo,X)
```

```
ff = convolute_ff(ff) #return input oject with ff$FCfit included

#the convolutions correlation to the feature contribution
for(i in 1:6) print(cor(ff$FCmatrix[,i],ff$FCfit[,i])^2)

#plotting the feature contributions
pars=par(no.readonly=TRUE) #save graphicals
par(mfrow=c(3,2),mar=c(2,2,2,2))
for(i in 1:6) {
  plot(ff$X[,i],ff$FCmatrix[,i],col="#00000030",ylim=range(ff$FCmatrix))
  points(ff$X[,i],ff$FCfit[,i],col="red",cex=0.2)
}
par(pars) #restore graphicals

## End(Not run)
```

---

`convolute_ff2` *Low-level function to estimate a specific set of feature contributions by corresponding features with kknn-package. Used to estimate goodness-of-fit of surface in show3d.*

---

## Description

Low-level function to estimate a selected combination feature contributions as function of selected features with leave-one-out k-nearest neighbor.

## Usage

```
convolute_ff2(ff,
              Xi,
              FCi = NULL,
              k.fun=function() round(sqrt(n.obs)/2),
              userArgs.kknn = alist(kernel="gaussian")            )
```

## Arguments

| | |
|---|---|
| `ff` | forestFloor object class "forestFloor_regression" or "forestFloor_multiClass" consisting of at least ff$X and ff$FCmatrix with two matrices of equal size |
| `Xi` | integer vector, of column indices of ff$X to estimate by. |
| `FCi` | integer vector, column indices of features contributions in ff$FCmatrix to estimate. If more than one , these columns will be summed by samples/rows. If NULL then FCi will match Xi. |
| `k.fun` | function to define k-neighbors to consider. n.obs is a constant as number of observations in ff$X. Hereby k neighbors is defined as a function k.fun of n.obs. To set k to a constant use e.g. k.fun = function() 10. k can also be overridden with userArgs.kknn = alist(kernel="Gaussian",kmax=10). |
| `userArgs.kknn` | argument list passed to train.kknn function for each convolution, see `train.kknn`. Arguments in this list have priority of any arguments passed by default by this wrapper function. See argument merger `train.kknn` |

## Details

convolute_ff2 is a wrapper of train.kknn to estimate feature contributions by a set of features. This function is e.g. used to estimate the visualized surface layer in show3d function. LOO CV is used to quantify how much of a feature contribution variation can by explained by a given surface. Can in theory also be used to quantify higher dimensional interaction effects, but randomForest do not learn much 3rd order (or higher) interactions. Do not support orderByImportance, thus Xi and FCi points to column order of training matrix X.

## Value

an numeric vector with one estimated feature contribution for any observation

## Author(s)

Soren Havelund Welling

## Examples

```
## Not run:
library(forestFloor)
library(randomForest)
library(rgl)
#simulate data
obs=2500
vars = 6
X = data.frame(replicate(vars,rnorm(obs)))
Y = with(X, X1^2 + 2*sin(X2*pi) + 8 * X3 * X4)
Yerror = 15 * rnorm(obs)
cor(Y,Y+Yerror)^2  #relatively noisy system
Y= Y+Yerror

#grow a forest, remeber to include inbag
rfo=randomForest(X,Y,keep.inbag=TRUE,ntree=1000,sampsize=800)

#obtain
ff = forestFloor(rfo,X)

#convolute the interacting feature contributions by their feature to understand relationship
fc34_convoluted = convolute_ff2(ff,Xi=3:4,FCi=3:4,  #arguments for the wrapper
                userArgs.kknn = alist(kernel="gaussian",k=25)) #arguments for train.kknn

#plot the joined convolution
plot3d(ff$X[,3],ff$X[,4],fc34_convoluted,
       main="convolution of two feature contributions by their own vaiables",
       #add some colour gradients to ease visualization
       #box.outliers squese all observations in a 2 std.dev box
       #univariately for a vector or matrix and normalize to [0;1]
       col=rgb(.7*box.outliers(fc34_convoluted),
               .7*box.outliers(ff$X[,3]),
               .7*box.outliers(ff$X[,4]))
       )
```

```
## End(Not run)
```

---

convolute_grid *Model structure grid estimated by feature contributions*

---

## Description

Low-level n-dimensional grid wrapper of kknn (not train.kknn). Predicts a grid structure on the basis of estimated feature contributions. Is used to draw one 2D surface in a 3D plot (show3d) on basis of feature contributions.

## Usage

```
convolute_grid          (ff,
                         Xi,
                         FCi = NULL,
                         grid = 30,
                         limit = 3,
                         zoom = 3,
                         k.fun=function() round(sqrt(n.obs)/2),
                         userArgs.kknn = alist(kernel="gaussian") )
```

## Arguments

| | |
|---|---|
| ff | the forestFloor object of class "forestFloor_regression" or "forestFloor_multiClass" at least containing ff$X and ff$FCmatrix with two matrices of equal size |
| Xi | the integer vector, of col indices of ff$X to estimate by, often of length 2 or 3. Note total number of predictions is a equal grid^"length of this vector". |
| FCi | the integer vector, of col indices of ff$FCmatrix. Those feature contributions to combine(sum) and estimate. If FCi=NULL, will copy Xi vector, which is the trivial choice. |
| grid | Either, an integer describing the number of grid.lines in each dimension(trivial choice) or, a full defined matrix of any grid position as defined by this function. |
| limit | a numeric scalar, number of standard deviations away from mean by any dimension to disregard outliers when spanning observations with grid. Set to limit=Inf outliers never should be disregarded. |
| zoom | numeric scalar, the size of the grid compared to the uni-variate range of data. If zoom=2 the grid will by any dimension span the double range of the observations. Outliers are disregarded with limit argument. |
| k.fun | function to define k-neighbors to consider. n.obs is a constant as number of observations in ff$X. Hereby k neighbors is defined as a function k.fun of n.obs. To set k to a constant use e.g. k.fun = function() 10. k can also be overridden with userArgs.kknn = alist(kernel="Gaussian",kmax=10). |
| userArgs.kknn | argument list to pass to train.kknn function for each convolution, see kknn for possible args. Arguments in this list will have priority of any passed by default by this wrapper function, see argument merger append.overwrite.alists |

## Details

This low-level function predicts feature contributions in a grid with train.kknn which is k-nearest neighbor + Gaussian weighting. This wrapper is used to construct the transparent grey surface in show3d.

## Value

a data frame, 1 + X variable columns. First column is the predicted summed feature contributions as a function of the following columns feature coordinates.

## Author(s)

Soren Havelund Welling

## Examples

```
## Not run:
## avoid testing of rgl 3D plot on headless non-windows OS
## users can disregard this sentence.
if(!interactive() && Sys.info()["sysname"]!="Windows") skip=TRUE

library(rgl)
library(randomForest)
library(forestFloor)

#simulate data
obs=1500
vars = 6
X = data.frame(replicate(vars,runif(obs)))*2-1
Y = with(X, X1*2 + 2*sin(X2*pi) + 3* (X3+X2)^2 )
Yerror = 1 * rnorm(obs)
var(Y)/var(Y+Yerror)
Y= Y+Yerror

#grow a forest, remember to include inbag
rfo=randomForest::randomForest(X,Y,
                               keep.inbag=TRUE,
                               ntree=1000,
                               replace=TRUE,
                               sampsize=500,
                               importance=TRUE)

#compute ff
ff = forestFloor(rfo,X)

#print forestFloor
print(ff)

#plot partial functions of most important variables first
Col=fcol(ff,1)
plot(ff,col=Col,orderByImportance=TRUE)
```

```
#the pure feature contributions
rgl::plot3d(ff$X[,2],ff$X[,3],apply(ff$FCmatrix[,2:3],1,sum),
            #add some colour gradients to ease visualization
            #box.outliers squese all observations in a 2 std.dev box
            #univariately for a vector or matrix and normalize to [0;1]
            col=fcol(ff,2,orderByImportance=FALSE))

#add grid convolution/interpolation
#make grid with current function
grid23 = convolute_grid(ff,Xi=2:3,userArgs.kknn= alist(k=25,kernel="gaus"),grid=50,zoom=1.2)
#apply grid on 3d-plot
rgl::persp3d(unique(grid23[,2]),unique(grid23[,3]),grid23[,1],alpha=0.3,
col=c("black","grey"),add=TRUE)
#anchor points of grid could be plotted also
rgl::plot3d(grid23[,2],grid23[,3],grid23[,1],alpha=0.3,col=c("black"),add=TRUE)

## and we se that their is almost no variance out of the surface, thus is FC2 and FC3
## well explained by the feature context of both X3 and X4

### next example show how to plot a 3D grid + feature contribution
## this 4D application is very experimental

#Make grid of three effects, 25^3 = 15625 anchor points
grid123 = convolute_grid(ff,
                         Xi=c(1:3),
                         FCi=c(1:3),
                         userArgs.kknn = alist(
                           k= 100,
                           kernel = "gaussian",
                           distance = 1),
                         grid=25,
                         zoom=1.2)

#Select a dimension to place in layers
uni2 = unique(grid123[,2])  #2 points to X1 and FC1
uni2=uni2[c(7,9,11,13,14,16,18)] #select some layers to visualize

## plotting any combination of X2 X3 in each layer(from red to green) having different value of X1
count = 0
add=FALSE
for(i in uni2) {
  count = count +1
  this34.plane = grid123[grid123[,2]==i,]
  if (count==2) add=TRUE

  #  plot3d(ff$X[,1],ff$X[,2]
  persp3d(unique(this34.plane[,3]),
          unique(this34.plane[,4]),
          this34.plane[,1], add=add,
          col=rgb(count/length(uni2),1-count/length(uni2),0),alpha=0.1)
}
```

```
## plotting any combination of X1 X3 in each layer(from red to green) having different value of X2
uni3 = unique(grid123[,4])  #2 points to X1 and FC1
uni3=uni3[c(7,9,11,13,14,16,18)] #select some layers to visualize
count = 0
add=FALSE
for(i in uni3) {
  count = count +1
  this34.plane = grid123[grid123[,4]==i,]
  if (count==2) add=TRUE


  #plot3d(ff$X[,1],ff$X[,2])
  persp3d(unique(this34.plane[,2]),
          unique(this34.plane[,3]),
          this34.plane[,1], add=add,
          col=rgb(count/length(uni3),1-count/length(uni3),0),alpha=0.1)

}

## End(Not run)
```

---

`fcol` *Generic colour module for forestFloor objects*

---

## Description

This colour module colour observations by selected variables. PCA decomposes a selection more than three variables. Space can be inflated by random forest variable importance, to focus coloring on influential variables. Outliers(>3std.dev) are automatically suppressed. Any colouring can be modified.

## Usage

```
fcol(ff, cols = NULL, orderByImportance = NULL, plotTest=NULL, X.matrix = TRUE,
     hue = NULL, saturation = NULL, brightness = NULL,
     hue.range  = NULL, sat.range  = NULL, bri.range  = NULL,
     alpha = NULL, RGB = NULL, byResiduals=FALSE, max.df=3,
     imp.weight = NULL, imp.exp = 1,outlier.lim = 3,RGB.exp=NULL)
```

## Arguments

| | |
|---|---|
| `ff` | a object of class "forestFloor_regression" or "forestFloor_multiClass" or a matrix or a data.frame. No missing values. X.matrix must be set TRUE for "forestFloor_multiClass" as colouring by multiClass feature contributions is not supported. |

cols — vector of indices of columns to colour by, will refer to ff$X if X.matrix=T and else ff$FCmatrix. If ff itself is a matrix or data.frame, indices will refer to these columns. If NULL, then all cols are selected.

orderByImportance — logical, should cols refer to X column order or columns sorted by variable importance. Input must be of forestFloor -class to use this. Set to FALSE if no importance sorting is wanted. Otherwise leave as is. If NULL, then TRUE for forestFloor objects and FALSE for data.frames and matrices.

plotTest — NULL(plot by test set if available), TRUE(plot by test set), FALSE(plot by train), "andTrain"(plot by both test and train)

X.matrix — logical, TRUE will use feature matrix, FALSE will use feature contribution matrix. Only relevant if ff input is forestFloor object.

hue — value within [0,1], hue=1 will be exactly as hue = 0 colour wheel settings, will skew the colour of all observations without changing the contrast between any two given observations. NULL is auto.

saturation — value within [0,1], mean saturation of colours, 0 is grey tone and 1 is maximal colourful. NULL is auto.

brightness — value within [0,1], mean brightness of colours, 0 is black and 1 is lightly colours. NULL is auto.

hue.range — value within [0,1], ratio of colour wheel, small value is small slice of colour wheel those little variation in colours. 1 is any possible colour except for RGB colour system. NULL is auto.

sat.range — value within [0,1], for colouring of 2 or more variables, a range of saturation is needed to obtain more degrees of freedom in the colour system. But as saturation of is preferred to be >.75 the range of saturation cannot here exceed .5. If NULL sat.range will set widest possible without exceeding range.

bri.range — value within [0,1], for colouring of 3 or more variables, a range of brightness is needed to obtain more degrees of freedom in the colour system. But as brightness of is preferred to be >.75 the range of saturation cannot here exceed .5. If NULL bri.range will set widest possible without exceeding range.

alpha — value within [0;1] transparency of colours. NULL is auto. The more points the less alpha, to avoid overplotting.

RGB — logical TRUE/FALSE,
RGB=NULL: will turn TRUE if one and only one variable is selected by cols input RGB=TRUE: "Red-Green-Blue": A non-linaer mapping from data to hsv. The gradient span from red to green to blue. Can still be altered by hue, saturation, brightness etc.
RGB=FALSE: "True-colour-system": Linear mapping from data to hsv colours. Sometimes more confusing, due to low contrast and over/under saturation.

byResiduals — logical, should colour gradient span the residuals of main effect fit(overrides X.matrix=). If no fit has been computed "is.null(ff$FCfit)", a temporarily main effect fit will be computed. Use ff = convolute_ff(ff) to only compute once and/or to modify fit parameters.

| | |
|---|---|
| max.df | integer 1, 2, or 3 only. Only for true-colour-system, the maximal allowed degrees of freedom (dimensionality) in a colour gradient. If more variables selected than max.df, PCA decomposes to requested degrees of freedom. max.df = 1 will give more simple colour gradients. |
| imp.weight | Logical?, Should importance from a forestFloor object be used to weight selected variables? Not possible if input ff is a matrix or data.frame. If randomForest(importance=TRUE) during training, variable importance will be used. Otherwise the more unreliable gini_importance coefficient. |
| imp.exp | exponent to modify influence of imp.weight. 0 is no influence. -1 is counter influence. 1 is linear influence. .5 is square root influence etc.. |
| outlier.lim | number from 0 to Inf. Any observation which univariately exceed this limit will be suppressed, as if it actually where on this limit. Normal limit is 3 standard deviations. Extreme outliers can otherwise reserve alone a very large part of a given linear colour gradient. This leads to visualization where outlier have one colour and any other observation have one single other colour. |
| RGB.exp | value between ]1;>1]. Defines steepness of the gradient of the RGB colour system Close to one green middle area is missing. For values higher than 2, green area is dominating |

## Details

fcol() is designed for the lazy programmer who likes to visualize a data set and/or a model-mapping swiftly without typing to much, check the examples for the typical use. To extra visual dimensions with colour gradients is useful to discover multivariate interactions in your randomForest model. fcol() both works for forestFloor, but also for any data.frame and matrix in general. Whenever an input parameter is not set or set to NULL, fcol() should choose a reasonble auto setting depending on possible other settings. In general if one manually set a parameter to something else than NULL it will override any possible auto setting.

## Value

a character vector specifying the colour of any observations. Each elements is something like "#F1A24340", where F1 is the hexadecimal of the red colour, then A2 is the green, then 43 is blue and 40 is transparency.

## Author(s)

Soren Havelund Welling

## Examples

```
## Not run:
#example 1 - fcol used on data.frame or matrix
library(forestFloor)
X = data.frame(matrix(rnorm(1000),nrow=1000,ncol=4))
X[] = lapply(X,jitter,amount = 1.5)

#single variable gradient by X1 (Unique colour system)
plot(X,col=fcol(X,1))
```

```
#double variable gradient by X1 and X2 (linear colour system)
plot(X,col=fcol(X,1:2))
#triple variable gradient (PCA-decomposed, linear colour system)
plot(X,col=fcol(X,1:3))
#higher based gradient    (PCA-decomposed, linear colour system)
plot(X,col=fcol(X,1:4))


#force linear col + modify colour wheel
plot(X,col=fcol(X,
                cols=1, #colouring by one variable
                RGB=FALSE,
                hue.range = 4, #cannot exceed 1, if colouing by more than one var
                               #except if max.df=1 (limits to 1D gradient)
                saturation=1,
                brightness = 0.6))

#colour by one dimensional gradient first PC of multiple variables
plot(X,col=fcol(X,
                cols=1:2, #colouring by multiple
                RGB=TRUE, #possible because max.df=1
                max.df = 1, #only 1D gradient (only first principal component)
                hue.range = 2, #can exceed 1, because max.df=1
                saturation=.95,
                brightness = 0.8))

##example 2 - fcol used with forestFloor objects
library(forestFloor)
library(randomForest)

X = data.frame(replicate(6,rnorm(1000)))
y = with(X,.3*X1^2+sin(X2*pi)+X3*X4)
rf = randomForest(X,y,keep.inbag = TRUE,sampsize = 400)
ff = forestFloor(rf,X)

#colour by most important variable
plot(ff,col=fcol(ff,1))

#colour by first variable in data set
plot(ff,col=fcol(ff,1,orderByImportance = FALSE),orderByImportance = FALSE)

#colour by feature contributions
plot(ff,col=fcol(ff,1:2,order=FALSE,X.matrix = FALSE,saturation=.95))

#colour by residuals
plot(ff,col=fcol(ff,3,orderByImportance = FALSE,byResiduals = TRUE))

#colour by all features (most useful for colinear variables)
plot(ff,col=fcol(ff,1:6))

#disable importance weighting of colour
#(important colours get to define gradients more)
plot(ff,col=fcol(ff,1:6,imp.weight = FALSE)) #useless X5 and X6 appear more colourful
```

```
#insert outlier in data set in X1 and X2
ff$X[1,1] = 10; ff$X[1,2] = 10

plot(ff,col=fcol(ff,1)) #colour not distorted, default: outlier.lim=3
plot(ff,col=fcol(ff,1,outlier.lim = Inf)) #colour gradient distorted by outlier
plot(ff,col=fcol(ff,1,outlier.lim = 0.5)) #too little outlier.lim


## End(Not run)
```

---

`forestFloor` *Compute out-of-bag cross-validated feature contributions to visualize model structures of randomForest models.*

---

## Description

Computes a cross validated feature contribution matrix from a randomForest model-fit and outputs a forestFloor S3 class object (a list), including unscaled importance and the original training set. The output object is the basis for all visualizations.

## Usage

```
forestFloor(rf.fit, X, Xtest=NULL, calc_np = FALSE, binary_reg = FALSE,
            bootstrapFC = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `rf.fit` | rf.fit, a random forest object as the output from randomForest::randomForest Train objects from caret::train is also experimentally supported, see how in the last use example |
| `X` | data.frame of input variables, numeric(continuous), discrete(treated as continuous) or factors(categorical). n_rows observations and n_columns features. X MUST be the exact same data.frame as used to train the random forest, see above item. If rows are re-ordered, the observations will not match the inbag-matrix resulting in distorted feature contributions. |
| `Xtest` | data.frame of input variables, numeric(continuous), discrete(treated as continuous) or factors(categorical). n_rows test_examples and n_columns features Xtest MUST have same number and order of columns(variables) as X. Number of rows and order can vary. |
| `calc_np` | TRUE/FALSE. Calculate Node Predictions(TRUE) or reuse information from rf.fit(FALSE)? Slightly faster when FALSE for regression. calc_np=TRUE will only take effect for rf.fit of class "randomForest" and type="regression". This option, is only for developmental purposes. Just set =FALSE always, as function will override this choice if not appropriate. |
| `binary_reg` | boolean, if TRUE binary classification can be changed to "percentage votes" of class 1, and thus be treated as regression. |

`bootstrapFC` boolean, if TRUE an extra column is added to FCmatrix or one extra matrix to FCarray accounting for the minor feature contributions attributed to random bootstraps or stratifications. Mainly useful to check FC row sums actually are equal to OOB-CV predictions, or to tweak randomForest into a "probability forest"-like model.

`...` For classification it is possible to manually set majorityTerminal=FALSE. For the randomForest classification implementation majorityTerminal is by default set to TRUE, as each tree uses majority vote within terminal nodes. In other implemenations terminal nodes are not neccesarily reduced by majority voting before aggregetion on ensemble level.

majorityTerminal, does not apply to random forest regressions.

impType = 1 or 2 and impScale = T / F and impClass = "factor level name" can modify what type importance to export from randomForest object to forestFloor object. imp-arguments are passed to `importance`

## Details

forestFloor computes out-of-bag cross validated feature contributions for a "randomForest" class object. Other packages will be supported in future, mail me a request. forestFloor guides you to discover the structure of a randomForest model fit. Check examples of how latent interactions can be identified with colour gradients.

What is FC, feature cotributions?: First, a local increment is the change of node prediction from parent to daughter node split by a given feature. Feature Contributions are the sums over all local increments for each observation for each feature divided by the number of trees. Thus a feature contribution summarizes the average outcome for all those times a given observation was split by a given feature. forestFloor use the inbag observations to calculate local increments, but only sum local increments over out-of-bag observations divided with OOBtimes. OOBtimes is the number of times a given observation have been out-of-bag, which is around ntrees / 3. In practice this removes a substantial self-leverage of observations to the corresponding feature contributions. Hereby visualizations becomes less noisy.

What is FC used for?: Feature contributions is smart way to decompose a RF mapping structure into additive components. Plotting FC's against variables values yields at first glance plots similar to marginal-effect plots, partial dependence plots and vector effect characteristic plots. This package forestFloor, makes use of feature contributions to separate main effects and identify plus quantify interactions. The advantages of forestFloor over typical partial.dependence plots are: (1) Easier to identify interactions. (2) feature contributions do not extrapolate the training observations as partial.dependence will do for correlated features. (3) The "goodness of visualization" (how exactly the plot represent the higher dimensional model structure) can be quantified. (4) Cheerful colours and 3D graphics thanks to the rgl package.

RF regression takes input features and outputs a target value. RF classification can output a pseudo probability vector with predicted class probability for each prediction. The RF mapping dimensionality for classification is different than for regression, as the output is no longer a scalar. The output is a vector with predicted class probability for each class. For binary classification this mapping can be simplified to a regression-like scalar as the probability of class_1 = 1 - class_2. Set binary_reg=TRUE for a binary RF classification to get regression like visualizations. For multi-class the output space is a probability space where any point is a probability prediction of each target class.

To plot forestFloor objects use plot-method plot.forestFloor and function show3d. Input parameters for classification or regression are not entirely the same. Check help-file plot.forestFloor and show3d. For 3-class problems the special function plot_simplex3 can plot the probability predictions in a 2D phase diagram (K-1 simplex).

## Value

the forestFloor function outputs(depending on type rf.fit) an object of either class "forestFloor_regression" or "forestFloor_multiClass" with following elements:

| | |
|---|---|
| X | a copy of the training data or feature space matrix/data.frame, X. The copy is passed unchanged from the input of this function. X is used in all visualization to expand the feature contributions over the features of which they were recorded. |
| Y | a copy of the target vector, Y. |
| importance | The gini-importance or permutation-importance a.k.a variable importance of the random forest object (unscaled). If rfo=randomForest(X,Y,importance=FALSE), gini-importance is used. Gini-importance is less reproducible and more biased. The extra time used to compute permutation-importance is negligible. |
| imp_ind | the importance indices is the order to sort the features by descending importance. imp_ind is used by plotting functions to present most relevant feature contributions first. If using gini-importance, the order of plots is more random and will favor continuous variables. The plots themselves will not differ. |
| FC_matrix | [ONLY forestFloor_regression.] feature contributions in a matrix.<br>n_row observations and n_column features - same dimensions as X. |
| FC_array | [ONLY forestFloor_multiClass.] feature contributions in a array.<br>n_row observations and n_column features and n_layer classes. First two dimensions will match dimensions of X. |

## Note

for more information goto forestFloor.dk

## Author(s)

Soren Havelund Welling

## References

Interpretation of QSAR Models Based on Random Forest Methods, http://dx.doi.org/10.1002/minf.201000173
Interpreting random forest classification models using a feature contribution method, http://arxiv.org/abs/1312.1121

## See Also

plot.forestFloor, show3d,

## Examples

```
## Not run:
## avoid testing of rgl 3D plot on headless non-windows OS
## users can disregard this sentence.
if(!interactive() && Sys.info()["sysname"]!="Windows") skipRGL=TRUE

#1 - Regression example:
set.seed(1234)
library(forestFloor)
library(randomForest)

#simulate data y = x1^2+sin(x2*pi)+x3*x4 + noise
obs = 5000 #how many observations/samples
vars = 6   #how many variables/features
#create 6 normal distr. uncorr. variables
X = data.frame(replicate(vars,rnorm(obs)))
#create target by hidden function
Y = with(X, X1^2 + sin(X2*pi) + 2 * X3 * X4 + 0.5 * rnorm(obs))

#grow a forest
rfo = randomForest(
  X, #features, data.frame or matrix. Recommended to name columns.
  Y, #targets, vector of integers or floats
  keep.inbag = TRUE,  # mandatory,
  importance = TRUE,  # recommended, else ordering by giniImpurity (unstable)
  sampsize = 1500 ,   # optional, reduce tree sizes to compute faster
  ntree = if(interactive()) 500 else 50 #speedup CRAN testing
)

#compute forestFloor object, often only 5-10% time of growing forest
ff = forestFloor(
  rf.fit = rfo,       # mandatory
  X = X,              # mandatory
  calc_np = FALSE,    # TRUE or FALSE both works, makes no difference
  binary_reg = FALSE  # takes no effect here when rfo$type="regression"
)

#print forestFloor
print(ff) #prints a text of what an 'forestFloor_regression' object is
plot(ff)

#plot partial functions of most important variables first
plot(ff,                       # forestFloor object
     plot_seq = 1:6,           # optional sequence of features to plot
     orderByImportance=TRUE    # if TRUE index sequence by importance, else by X column
)

#Non interacting features are well displayed, whereas X3 and X4 are not
#by applying color gradient, interactions reveal themself
#also a k-nearest neighbor fit is applied to evaluate goodness-of-fit
Col=fcol(ff,3,orderByImportance=FALSE) #create color gradient see help(fcol)
plot(ff,col=Col,plot_GOF=TRUE)
```

```
#feature contributions of X3 and X4 are well explained in the context of X3 and X4
# as GOF R^2>.8

show3d(ff,3:4,col=Col,plot_GOF=TRUE,orderByImportance=FALSE)

#if needed, k-nearest neighbor parameters for goodness-of-fit can be accessed through convolute_ff
#a new fit will be calculated and saved to forstFloor object as ff$FCfit
ff = convolute_ff(ff,userArgs.kknn=alist(kernel="epanechnikov",kmax=5))
plot(ff,col=Col,plot_GOF=TRUE) #this computed fit is now used in any 2D plotting.


###
#2 - Multi classification example:   (multi is more than two classes)
set.seed(1234)
library(forestFloor)
library(randomForest)

data(iris)
X = iris[,!names(iris) %in% "Species"]
Y = iris[,"Species"]

rf = randomForest(
  X,Y,
  keep.forest=TRUE,  # mandatory
  keep.inbag=TRUE,   # mandatory
  samp=20,           # reduce complexity of mapping structure, with same OOB%-explained
  importance  = TRUE # recommended, else ordering by giniImpurity (unstable)
)

ff = forestFloor(rf,X)

plot(ff,plot_GOF=TRUE,cex=.7,
     colLists=list(c("#FF0000A5"),
                   c("#00FF0050"),
                   c("#0000FF35")))

#...and 3D plot, see show3d
show3d(ff,1:2,1:2,plot_GOF=TRUE)

#...and simplex plot (only for three class problems)
plot_simplex3(ff)
plot_simplex3(ff,zoom.fit = TRUE)

#...and 3d simplex plots (rough look, Z-axis is feature)
plot_simplex3(ff,fig3d = TRUE)

###
#3 - binary regression example
#classification of two classes can be seen as regression in 0 to 1 scale
set.seed(1234)
library(forestFloor)
library(randomForest)
```

```
data(iris)
X = iris[-1:-50,!names(iris) %in% "Species"] #drop third class virginica
Y = iris[-1:-50,"Species"]
Y = droplevels((Y)) #drop unused level virginica

rf = randomForest(
  X,Y,
  keep.forest=TRUE,  # mandatory
  keep.inbag=TRUE,   # mandatory
  samp=20,           # reduce complexity of mapping structure, with same OOB%-explained
  importance  = TRUE # recommended, else giniImpurity
)

ff = forestFloor(rf,X,
                 calc_np=TRUE,    #mandatory to recalculate
                 binary_reg=TRUE) #binary regression, scale direction is printed
Col = fcol(ff,1) #color by most important feature
plot(ff,col=Col)   #plot features

#interfacing with rgl::plot3d
show3d(ff,1:2,col=Col,plot.rgl.args = list(size=2,type="s",alpha=.5))


##example with caret

rm(list=ls())
library(caret)
library(forestFloor)
N = 1000
vars = 15
noise_factor = .3
bankruptcy_baserate = 0.2
X = data.frame(replicate(vars,rnorm(N)))
y.signal = with(X,X1^2+X2^2+X3*X4+X5+X6^3+sin(X7*pi)*2) #some non-linear f
y.noise = rnorm(N) * sd(y.signal) * noise_factor
y.total = y.noise+y.signal
y = factor(y.total>=quantile(y.total,1-bankruptcy_baserate))

set.seed(1)
caret_train_obj <- train(
  x = X, y = y,
  method = "rf",
  keep.inbag = TRUE,  #always set keep.inbag=TRUE passed as ... parameter to randomForest
  ntree=50, #speed up this example, if set too low, forestFloor will fail
 )

rf = caret_train_obj$finalModel #extract model
if(!all(rf$y==y)) warning("seems like training set have been resampled, using smote?")
ff = forestFloor(rf,X,binary_reg = T)

#... or simply pass train
ff = forestFloor(caret_train_obj,X,binary_reg = T)
plot(ff,1:6,plot_GOF = TRUE)
```

```
## End(Not run)
```

---

`importanceExportWrapper` *Importance Export Wrapper (internal)*

---

## Description

wrapping randomForest::importance and check if one and only one importance is exported

## Usage

```
importanceExportWrapper(rf, type = NULL, class = NULL, scale = NULL)
```

## Arguments

| | |
|---|---|
| `rf` | object of class randomForest |
| `type` | type of importance, 1 permutation based, 2 loss function based (gini/least squares) |
| `class` | character of factor level to get class specific importance, only for type 1 importance for classification random forest |
| `scale` | TRUE FALSE or NULL, passed to randomForest::importance |

## Details

internal wrapper to get importance

## Value

vector of importance for each feature

## Author(s)

Soren Havelund Welling, 2017

## See Also

importance

## Examples

```
#no examples only for internal use
```

plot.forestFloor *plot.forestFloor_regression*

## Description

A method to plot an object of forestFloor-class. Plot partial feature contributions of the most important variables. Colour gradients can be applied to show possible interactions. Fitted function(plot_GOF) describe FC only as a main effect and quantifies 'Goodness Of Fit'.

## Usage

```
## S3 method for class 'forestFloor_regression'
 plot(
  x,
  plot_seq=NULL,
  plotTest = NULL,
  limitY=TRUE,
  orderByImportance=TRUE,
  cropXaxes=NULL,
  crop_limit=4,
  plot_GOF = TRUE,
  GOF_args = list(col="#33333399"),
  speedup_GOF = TRUE,
  ...)

## S3 method for class 'forestFloor_multiClass'
 plot(
  x,
  plot_seq = NULL,
  label.seq = NULL,
  plotTest  = NULL,
  limitY = TRUE,
  col = NULL,
  colLists = NULL,
  orderByImportance = TRUE,
  fig.columns = NULL,
  plot_GOF = TRUE,
  GOF_args = list(),
  speedup_GOF = TRUE,
  jitter_these_cols = NULL,
  jitter.factor = NULL,
  ...)
```

## Arguments

| | |
|---|---|
| x | forestFloor-object, also abbreviated ff. Basically a list of class="forestFloor" containing feature contributions, features, targets and variable importance. |

| | |
|---|---|
| plot_seq | a numeric vector describing which variables and in what sequence to plot. Ordered by importance as default. If orderByImportance = F, then by feature/column order of training data. |
| label.seq | [only classification] a numeric vector describing which classes and in what sequence to plot. NULL is all classes ordered is in levels in x$Y of forestFloor_mulitClass object x. |
| plotTest | NULL(plot by test set if available), TRUE(plot by test set), FALSE(plot by train), "andTrain"(plot by both test and train) |
| fig.columns | [only for multiple plotting], how many columns per page. default(NULL) is 1 for one plot, 2 for 2, 3 for 3, 2 for 4 and 3 for more. |
| limitY | TRUE/FLASE, constrain all Yaxis to same limits to ensure relevance of low importance features is not over interpreted |
| col | Either a colur vector with one colour per plotted class label or a list of colour vectors. Each element is a colour vector one class. Colour vectors in list are normally either of length 1 with or of length equal to number of training observations. NULL will choose standard one colour per class. |
| colLists | Deprecetated, will be replaced by col input |
| jitter_these_cols | vector to apply jitter to x-axis in plots. Will refer to variables. Useful to for categorical variables. Default=NULL is no jitter. |
| jitter.factor | value to decide how much jitter to apply. often between .5 and 3 |
| orderByImportance | TRUE / FALSE should plotting and plot_seq be ordered after importance. Most important feature plot first(TRUE) |
| cropXaxes | a vector of indices of which zooming of x.axis should look away from outliers |
| crop_limit | a number often between 1.5 and 5, referring limit in sigmas from the mean defining outliers if limit = 2, above selected plots will zoom to +/- 2 std.dev of the respective features. |
| plot_GOF | Boolean TRUE/FALSE. Should the goodness of fit be plotted as a line? |
| GOF_args | Graphical arguments fitted lines, see points for parameter names. |
| speedup_GOF | Should GOF only computed on reasonable sub sample of data set to speedup computation. GOF estimation leave-one-out-kNN becomes increasingly slow for +1500 samples. |
| ... | ... other arguments passed to par or plot<br>. e.g. mar=, mfrow=, is passed to par, and cex= is passed to plot. par() arguments are reset immediately as plot function returns. |

## Details

The method plot.forestFloor visualizes partial plots of the most important variables first. Partial dependence plots are available in the randomForest package. But such plots are single lines(1d-slices) and do not answer the question: Is this partial function(PF) a fair generalization or subject to global or local interactions.

## Author(s)

Soren Havelund Welling

## Examples

```
## Not run:
  ## avoid testing of rgl 3D plot on headless non-windows OS
  ## users can disregard this sentence.
  if(!interactive() && Sys.info()["sysname"]!="Windows") skipRGL=TRUE

  ###
  #1 - Regression example:
  set.seed(1234)
  library(forestFloor)
  library(randomForest)

  #simulate data y = x1^2+sin(x2*pi)+x3*x4 + noise
  obs = 5000 #how many observations/samples
  vars = 6   #how many variables/features
  #create 6 normal distr. uncorr. variables
  X = data.frame(replicate(vars,rnorm(obs)))
  #create target by hidden function
  Y = with(X, X1^2 + sin(X2*pi) + 2 * X3 * X4 + 0.5 * rnorm(obs))

  #grow a forest
  rfo = randomForest(
    X, #features, data.frame or matrix. Recommended to name columns.
    Y, #targets, vector of integers or floats
    keep.inbag = TRUE,  # mandatory,
    importance = TRUE,  # recommended, else ordering by giniImpurity (unstable)
    sampsize = 1500 ,   # optional, reduce tree sizes to compute faster
    ntree = if(interactive()) 1000 else 25 #speedup CRAN testing
  )

  #compute forestFloor object, often only 5-10% time of growing forest
  ff = forestFloor(
    rf.fit = rfo,       # mandatory
    X = X,              # mandatory
    calc_np = FALSE,    # TRUE or FALSE both works, makes no difference
    binary_reg = FALSE  # takes no effect here when rfo$type="regression"
  )

  #print forestFloor
  print(ff) #prints a text of what an 'forestFloor_regression' object is
  plot(ff)

  #plot partial functions of most important variables first
  plot(ff,                       # forestFloor object
       plot_seq = 1:6,           # optional sequence of features to plot
      orderByImportance=TRUE    # if TRUE index sequence by importance, else by X column
  )
```

```
#Non interacting features are well displayed, whereas X3 and X4 are not
#by applying color gradient, interactions reveal themself
#also a k-nearest neighbor fit is applied to evaluate goodness-of-fit
Col=fcol(ff,3,orderByImportance=FALSE) #create color gradient see help(fcol)
plot(ff,col=Col,plot_GOF=TRUE)

#feature contributions of X3 and X4 are well explained in the context of X3 and X4
# as GOF R^2>.8

show3d(ff,3:4,col=Col,plot_GOF=TRUE,orderByImportance=FALSE)

#if needed, k-nearest neighbor parameters for goodness-of-fit can be accessed through convolute_ff
#a new fit will be calculated and saved to forstFloor object as ff$FCfit
ff = convolute_ff(ff,userArgs.kknn=alist(kernel="epanechnikov",kmax=5))
plot(ff,col=Col,plot_GOF=TRUE) #this computed fit is now used in any 2D plotting.


###
#2 - Multi classification example:   (multi is more than two classes)
set.seed(1234)
library(forestFloor)
library(randomForest)

data(iris)
X = iris[,!names(iris) %in% "Species"]
Y = iris["Species"]

rf = randomForest(
  X,Y,
  keep.forest=TRUE,  # mandatory
  keep.inbag=TRUE,   # mandatory
  samp=20,          # reduce complexity of mapping structure, with same OOB%-explained
  importance  = TRUE, # recommended, else ordering by giniImpurity (unstable)
  ntree = if(interactive()) 1000 else 25 #speedup CRAN testing
)

ff = forestFloor(rf,X)

plot(ff,plot_GOF=TRUE,cex=.7,
  col=c("#FF0000A5","#00FF0050","#0000FF35") #one col per plotted class
)

#...and 3D plot, see show3d
show3d(ff,1:2,1:2,plot_GOF=TRUE)

#...and simplex plot (only for three class problems)
plot_simplex3(ff)
plot_simplex3(ff,zoom.fit = TRUE)

#...and 3d simplex plots (rough look, Z-axis is feature)
plot_simplex3(ff,fig3d = TRUE)

###
```

```
#3 - binary regression example
#classification of two classes can be seen as regression in 0 to 1 scale
set.seed(1234)
library(forestFloor)
library(randomForest)
data(iris)
X = iris[-1:-50,!names(iris) %in% "Species"] #drop third class virginica
Y = iris[-1:-50,"Species"]
Y = droplevels((Y)) #drop unused level virginica

rf = randomForest(
  X,Y,
  keep.forest=TRUE,  # mandatory
  keep.inbag=TRUE,   # mandatory
  samp=20,          # reduce complexity of mapping structure, with same OOB%-explained
  importance  = TRUE, # recommended, else giniImpurity
   ntree = if(interactive()) 1000 else 25 #speedup CRAN testing
)

ff = forestFloor(rf,X,
                 calc_np=TRUE,    #mandatory to recalculate
                 binary_reg=TRUE) #binary regression, scale direction is printed
Col = fcol(ff,1) #color by most important feature
plot(ff,col=Col)   #plot features

#interfacing with rgl::plot3d
show3d(ff,1:2,col=Col,plot.rgl.args = list(size=2,type="s",alpha=.5))

## End(Not run)
```

---

`plot_simplex3` *3-class simplex forestFloor plot*

---

## Description

3-class forestFloor plotted in a 2D simplex. The plot describes with feature contributions the change of predicted class probability for each sample due a single variable given all other variables. This plot is better than regular multiclass plots (plot.forestFloor_multiClass) to show the change of class probabilities, but the feature values can only be depcited as a colour gradient. But (fig3d=TRUE) allows the feature value to be depicted by the Z-axis as a extra pop-up 3D plot.

## Usage

```
plot_simplex3(ff,
              Xi           = NULL,
              includeTotal = TRUE,
              label.col    = NULL,
              fig.cols     = 3,
              fig.rows     = NULL,
```

```
auto.alpha   = 0.25,
fig3d        = FALSE,
restore_par  = TRUE,
set_pars     = TRUE,
zoom.fit     = NULL,
var.col      = NULL,
plot.sep.centroid = TRUE)
```

## Arguments

| | |
|---|---|
| ff | x also abbrivated ff, forestFloor_mulitClass the output from the forestFloor function. Must have 3 classes exactly. |
| Xi | vector of integer indices (refeering to column order of trainingset) to what feature contributions should be plotted in individual plots. |
| includeTotal | TRUE / FALSE. Combined separation of all feature contributions, which is equal to the separation of the entire model can be included. |
| label.col | a colour vector of K classes length defining the colour of each class for plotting. NULL is auto. |
| fig.cols | How many columns should be plotted sideways, is passed to par(mfrow=c(fig.rows,fig.cols)) |
| fig.rows | How many rows should be plotted, is passed to par(mfrow=c(fig.rows,fig.cols)) NULL is auto |
| auto.alpha | a scalar between 0.5 to 1 most often. Low values increase transparancy of points used to avoid overplotting. auto.alpha is alpha corrected of samplesize such that less adjustment is needed. |
| fig3d | TRUE/FALSE, a 3D plot including the variable as an axis can be co-plotted with rgl. |
| restore_par | TRUE/FALSE, calls to graphics par() will be reset |
| set_pars | TRUE/FALSE, if FALSE plot function will rather inherrit plot settings global pars. USeful for multi plotting loops. |
| zoom.fit | NULL/TRUE, if TRUE zooming on samples will be applied. Do not set to FALSE. |
| var.col | a single colour or a colour vector of N samples length. Samples will be coloured accordingly. use function fcol to make colour gradient e.g. by the variable values themselves. See example fcol. |
| plot.sep.centroid | TRUE/FALSE. Should the average bootstrap prediction be plotted? If no bootstrap stratification, the average bootstrap prediction is equal to class distribution training set. RF model probalistic predictions is equal to average bootstrap prediction plus all feature contributions. |

## Details

Random forest 3 class maps from a feature space to a 3 dimensional (K-1) probability simplex space, which can be plotted in 2D because class probabilities sum to one, and class feature contributions sum to zero. The centroid these plots is the prior of the random forest model. The prior, unless modified with statification is the target class distribution. Default majority voting lines would run from middle to the corners.

## Author(s)

Soren Havelund Welling

## Examples

```
## Not run:
  library(randomForest)
  library(forestFloor)
  require(utils)

  data(iris)

  X = iris[,!names(iris) %in% "Species"]
  Y = iris[,"Species"]
  as.numeric(Y)
  rf.test42 = randomForest(X,Y,keep.forest=TRUE,
    replace=FALSE,keep.inbag=TRUE,samp=15,ntree=100)
  ff.test42 = forestFloor(rf.test42,X,calc_np=FALSE,binary_reg=FALSE)

  plot(ff.test42,plot_GOF=TRUE,cex=.7,
       colLists=list(c("#FF0000A5"),
                     c("#00FF0050"),
                     c("#0000FF35")))

  show3d(ff.test42,1:2,3:4,plot_GOF=TRUE)

  #plot all effect 2D only
  pars = plot_simplex3(ff.test42,Xi=c(1:3),restore_par=FALSE,zoom.fit=NULL,
    var.col=NULL,fig.cols=2,fig.rows=1,fig3d=FALSE,includeTotal=TRUE,auto.alpha=.4
    ,set_pars=TRUE)

  pars = plot_simplex3(ff.test42,Xi=0,restore_par=FALSE,zoom.fit=NULL,
    var.col=alist(alpha=.3,cols=1:4),fig3d=FALSE,includeTotal=TRUE,
    auto.alpha=.8,set_pars=FALSE)

  for (I in ff.test42$imp_ind[1:4])  {
    #plotting partial OOB-CV separation(including interactions effects)
    #coloured by true class
    pars = plot_simplex3(ff.test42,Xi=I,restore_par=FALSE,zoom.fit=NULL,
    var.col=NULL,fig.cols=4,fig.rows=2,fig3d=TRUE,includeTotal=FALSE,label.col=1:3,
    auto.alpha=.3,set_pars = (I==ff.test42$imp_ind[1]))

    #coloured by varaible value
    pars = plot_simplex3(ff.test42,Xi=I,restore_par=FALSE,zoom.fit=TRUE,
    var.col=alist(order=FALSE,alpha=.8),fig3d=FALSE,includeTotal=(I==4),
    auto.alpha=.3,set_pars=FALSE)
  }

## End(Not run)
```

print.forestFloor *print summary of forestFloor.Object*

## Description

This function simply states the obvious and returns the elements inside the object list.

## Usage

```
## S3 method for class 'forestFloor_regression'
    print(x,...)
## S3 method for class 'forestFloor_multiClass'
print(x,...)
```

## Arguments

| | |
|---|---|
| x | x also abbreviated ff, forestFloor_Object the output from the forestFloor function |
| ... | ... other arguments passed to generic print function |

## Details

prints short help text for usage of a forestFloor_object

## Author(s)

Soren Havelund Welling

## Examples

```
## Not run:
#simulate data
obs=1000
vars = 6
X = data.frame(replicate(vars,rnorm(obs)))
Y = with(X, X1^2 + sin(X2*pi) + 2 * X3 * X4 + 0.5 * rnorm(obs))

#grow a forest, remeber to include inbag
rfo=randomForest::randomForest(X,Y,keep.inbag=TRUE)

#compute topology
ff = forestFloor(rfo,X)

#print forestFloor
print(ff)

## End(Not run)
```

recTree *recursiveTree: cross-validated feature contributions*

## Description

internal C++ functions to compute feature contributions for a random Forest

## Usage

```
recTree(  vars, obs, ntree, calculate_node_pred, X,Y,majorityTerminal, leftDaughter,
    rightDaughter, nodestatus, xbestsplit, nodepred, bestvar,
    inbag, varLevels, OOBtimes, localIncrements)

multiTree(vars, obs, ntree, nClasses,          X,Y,majorityTerminal, leftDaughter,
    rightDaughter, nodestatus, xbestsplit, nodepred, bestvar,
    inbag, varLevels, OOBtimes, localIncrements)
```

## Arguments

| | |
|---|---|
| vars | number of variables in X |
| obs | number of observations in X |
| ntree | number of trees starting from 1 function should iterate, cannot be higher than columns of inbag |
| nClasses | number of classes in classification forest |
| calculate_node_pred | should the node predictions be recalculated(true) or reused from nodepred-matrix(false & regression) |
| X | X training matrix |
| Y | target vector, factor or regression |
| majorityTerminal | bool, majority vote in terminal nodes? Default is FALSE for regression. Set only to TRUE when binary_reg=TRUE. |
| leftDaughter | a matrix from a the output of randomForest rf$forest$leftDaughter the node.number/row.number of the leftDaughter in a given tree by column |
| rightDaughter | a matrix from a the output of randomForest rf$forest$rightDaughter the node.number/row.number of the rightDaughter in a given tree by column |
| nodestatus | a matrix from a the output of randomForest rf$forest$nodestatus the nodestatus of a given node in a given tree |
| xbestsplit | a matrix from a the output of randomForest rf$forest$xbestsplit. The split point of numeric variables or the binary split of categorical variables. See help file of randomForest::getTree for details of binary expansion for categorical splits. |
| nodepred | a matrix from a the output of randomForest rf$forest$xbestsplit. The inbag target average for regression mode and the majority target class for classification |

| | |
|---|---|
| bestvar | a matrix from a the output of randomForest rf$forest$xbestsplit the inbag target average for regression mode and the majority target class for classification |
| inbag | a matrix as the output of randomForest rf$inbag. Contain counts of how many times a sample was selected for a given tree. |
| varLevels | the number of levels of all variables, 1 for continuous or discrete, >1 for categorical variables. This is needed for categorical variables to interpret binary split from xbestsplit. |
| OOBtimes | number of times a certain observation was out-of-bag in the forest. Needed to compute cross-validated feature contributions as these are summed local increments over out-of-bag observations over features divided by this number. In previous implementation(rfFC), articles(see references) feature contributions are summed by all observations and is divived by ntrees. |
| localIncrements | an empty matrix to store localIncrements during computation. As C++ function returns, the input localIncrement matrix contains the feature contributions. |

## Details

This is function is excuted by the function forestFloor. This is a c++/Rcpp implementation computing feature contributions. The main differences from this implementation and the rfFC-package(Rforge), is that these feature contributions are only summed over out-of-bag samples yields a cross-validation. This implementation allows sample replacement, binary and multi-classification.

## Value

no output, the feature contributions are writtten directly to localIncrements input

## Author(s)

Soren Havelund Welling

## References

Interpretation of QSAR Models Based on Random Forest Methods, http://dx.doi.org/10.1002/minf.201000173
Interpreting random forest classification models using a feature contribution method, http://arxiv.org/abs/1312.1121

---

# show3d *make forestFloor 3D-plot of random forest feature contributions*

---

## Description

2 features features(horizontal XY-plane) and one combined feature contribution (vertical Z-axis). Surface response layer will be estimated(kknn package) and plotted alongside the data points. 3D graphic device is rgl. Will dispatch methods show3d.forestFloor_regression for regression and show3d_forestFloor_multiClass for classification.

## Usage

```
## S3 method for class 'forestFloor_regression'
 show3d(
      x,
      Xi  = 1:2,
      FCi = NULL,
      col = "#12345678",
      plotTest = NULL,
      orderByImportance = TRUE,
      surface=TRUE,
      combineFC = sum,
      zoom=1.2,
      grid.lines=30,
      limit=3,
      cropPointsOutSideLimit = TRUE,
      kknnGrid.args = alist(),
      plot.rgl.args = alist(),
      surf.rgl.args = alist(),
      user.gof.args = alist(),
      plot_GOF = TRUE,
      ...)

## S3 method for class 'forestFloor_multiClass'
show3d(
      x,
      Xi,
      FCi=NULL,
      plotTest = NULL,
      label.seq=NULL,
      kknnGrid.args=list(NULL),
      plot.rgl.args=list(),
      plot_GOF=FALSE,
      user.gof.args=list(NULL),
      ...)
```

## Arguments

| | |
|---|---|
| x | forestFloor" class object |
| Xi | integer vector of length 2 indices of feature columns |
| FCi | integer vector of length 1 to p variables indices of feature contributions columns |
| col | a colour vector. One colour or colour palette(vector). |
| plotTest | NULL(plot by test set if available), TRUE(plot by test set), FALSE(plot by train), "andTrain"(plot by both test and train) |
| orderByImportance | should indices order by 'variable importance' or by matrix/data.frame order? |
| surface | should a surface be plotted also? |

`combineFC` a row function applied on selected columns(FCi) on $FCmatrix or $FCarray. How should feature contributions be combined? Default is `sum`.

`zoom` grid can be expanded in all directions by a factor

`grid.lines` how many grid lines should be used. Total surface anchor points in plot is grid.lines^2. May run slow above 200-500 depending on hardware.

`limit` a number. Sizing of grid does not consider outliers outside this limit of e.g. 3 SD deviations univariately.

`cropPointsOutSideLimit`
#if points exceed standard deviation limit, they will not be plotted

`kknnGrid.args` argument list, any possible arguments to kknnkknn
These default wrapper arguments can hereby be overwritten:
wrapper = alist( formula=fc~., # do not change
train=Data, # do not change
k=k, # integer < n_observations. k>100 may run slow.
kernel="gaussian", #distance kernel, other is e.g. kernel="triangular"
test=gridX #do not change
)
see kknnkknn to understand parameters. k is set by default automatically to a half times the square root of observations, which often gives a reasonable balance between robustness and adeptness. k neighbors and distance kernel can be changed be passing kknnGrid.args = alist(k=5,kernel="triangular",scale=FALSE), hereby will default k and default kernel be overwritten. Moreover the scale argument was not specified by this wrapper and therefore not conflicting, the argument is simply appended.

`plot.rgl.args` pass argument to rgl::plot3d, can override any argument of this wrapper, defines plotting space and plot points. See plot3d for documentation of graphical arguments.
wrapper_arg = alist( x=xaxis, #do not change, x coordinates
y=yaxis, #do not change, y coordinates
z=zaxis, #do not change, z coordinates
col=col, #colouring evaluated within this wrapper function
xlab=names(X)[1], #xlab, label for x axis
ylab=names(X)[2], #ylab, label for y axis
zlab=paste(names(X[,FCi]),collapse=" - "), #zlab, label for z axis
alpha=.4, #points transparency
size=3, #point size
scale=.7, #z axis scaling
avoidFreeType = T, #disable freeType=T plug-in. (Postscript labels)
add=FALSE #do not change, should graphics be added to other rgl-plot?
)

`surf.rgl.args` wrapper_arg = alist( x=unique(grid[,2]), #do not change, values of x-axis
y=unique(grid[,3]), #do not change, values of y-axis
z=grid[,1], #do not change, response surface values
add=TRUE, #do not change, surface added to plotted points
alpha=0.4 #transparency of surface, [0;1]
)

| | |
|---|---|
| | see rgl::persp3d for other graphical arguments notice the surface is added onto plotting of points, thus can e.g. labels not be changed from here. |
| label.seq | a numeric vector describing which classes and in what sequence to plot. NULL is all classes ordered is in levels in x$Y of forestFloor_mulitClass object x. |
| user.gof.args | argument list passed to internal function convolute_ff2, which can modify how goodness-of-fit is computed. Number of neighbors and kernel can be set manually with e.g. list(kmax=40,kernel="gaussian"). Default pars should work already in most cases. Function convolute_ff2 computed leave-one-out CV prediction the feature contributions from the chosen context of the visualization. |
| plot_GOF | Boolean TRUE/FALSE. Should the goodness of fit be computed and plotted is main of 3D plot? If false, no GOF input pars are useful. |
| ... | not used at the moment |

## Details

show3d plot one or more combined feature contributions in the context of two features with points representing each data point. The input object must be a "forestFloor_regression" or "forestFloor_multiClass" S3 class object , and should at least contain $X the data.frame of training data, $FCmatrix the feature contributions matrix. Usually this object are formed with the function forestFloor having a random forest model fit as input. Actual visualization differs for each class.

## Value

no value

## Author(s)

Soren Havelund Welling

## Examples

```
## Not run:
## avoid testing of rgl 3D plot on headless non-windows OS
## users can disregard this sentence.
if(!interactive() && Sys.info()["sysname"]!="Windows") skipRGL=TRUE

library(forestFloor)
library(randomForest)
#simulate data
obs=2500
vars = 6

X = data.frame(replicate(vars,rnorm(obs)))
Y = with(X, X1^2 + sin(X2*pi) + 2 * X3 * X4 + 1 * rnorm(obs))


#grow a forest, remeber to include inbag
rfo=randomForest(X,Y,keep.inbag = TRUE,sampsize=1500,ntree=500)

#compute topology
```

```
ff = forestFloor(rfo,X)


#print forestFloor
print(ff)

#plot partial functions of most important variables first
plot(ff)

#Non interacting functions are well displayed, whereas X3 and X4 are not
#by applying different colourgradient, interactions reveal themself
Col = fcol(ff,3)
plot(ff,col=Col)

#in 3D the interaction between X3 and X reveals itself completely
show3d(ff,3:4,col=Col,plot.rgl=list(size=5))

#although no interaction, a joined additive effect of X1 and X2
Col = fcol(ff,1:2,X.m=FALSE,RGB=TRUE) #colour by FC-component FC1 and FC2 summed
plot(ff,col=Col)
show3d(ff,1:2,col=Col,plot.rgl=list(size=5))

#...or two-way gradient is formed from FC-component X1 and X2.
Col = fcol(ff,1:2,X.matrix=TRUE,alpha=0.8)
plot(ff,col=Col)
show3d(ff,1:2,col=Col,plot.rgl=list(size=5))

## End(Not run)
```

---

`vec.plot` *Compute and plot vector effect characteristics for a given multivariate model*

---

## Description

vec.plot visualizes the vector effect characteristics of a given model. Geometrically it corresponds to a specific 2D or 3D slice of a higher dimensional mapping structure. One variable (2D plot) or two variables (3D plot) are screened within the range of the training data, while remaining variables are fixed at the univariate means (as default). If remaining variables do not interact strongly with plotted variable(s), vec.plot is a good tool to break up a high-dimensional model structure into separate components.

## Usage

```
vec.plot(model,X,i.var,grid.lines=100,VEC.function=mean,
         zoom=1,limitY=F,moreArgs=list(),...)
```

## Arguments

| | |
|---|---|
| model | model_object which has a defined method predict.model, which can accept arguments as showed for randomForest e.g. library(randomForest) model = randomForest(X,Y) predict(model,X) |
| | where X is the training features and Y is the training response vector(numeric) |
| X | matrix or data.frame being the same as input to model |
| i.var | vector, of column_numbers of variables to scan. No plotting is available for more than two variables. |
| grid.lines | scalar, number of values by each variable to be predicted by model. Total number of combinations = grid.lines^length(i_var). |
| VEC.function | function, establish one fixed value for any remaining variables(those not chosen by i.var). Default is to use the mean of variables. |
| zoom | scalar, number defining the size.factor of the VEC.surface compared to data range of scanned variables. Bigger number is bigger surface. |
| limitY | boolean, if TRUE Y-axis is standardized for any variable. Useful for composite plots as shown in example. |
| moreArgs | any lower level graphical args passed to rgl::surface3d or points depending on number of variables(length of i.var) |
| ... | any lower level graphical args passed to rgl::plot3d or plot depending on number of variables(length of i.var) |

## Details

vec.plot visualizes the vector effect characteristics of a given model. One(2D plot) or two(3D plot) variables are screened within the range of the training data, while remaining variables are fixed at the univariate means of each them(as default). If remaining variables do not interact strongly with plotted variable(s), vec.plot is a good tool to break up a high-dimensional model topology in separate components.

## Value

no value

## Author(s)

Soren Havelund Welling

## Examples

```
## Not run:
## avoid testing of rgl 3D plot on headless non-windows OS
## users can disregard this sentence.
if(!interactive() && Sys.info()["sysname"]!="Windows") skipRGL=TRUE
library(randomForest)
library(forestFloor)

#simulate data
```

```
obs=2000
vars = 6
X = data.frame(replicate(vars,rnorm(obs)))
Y = with(X, X1^2 + 2*sin(X2*pi) + 2 * X3 * (X4+.5))
Yerror = 1 * rnorm(obs)
var(Y)/var(Y+Yerror)
Y= Y+Yerror

#grow a forest, remeber to include inbag
rfo2=randomForest(X,Y,keep.inbag=TRUE,sampsize=800)

#plot partial functions of most important variables first
pars=par(no.readonly=TRUE) #save previous graphical paremeters
par(mfrow=c(2,3),mar=c(2,2,1,1))
for(i in 1:vars) vec.plot(rfo2,X,i,zoom=1.5,limitY=TRUE)
par(pars) #restore

#plot partial functions of most important variables first
for(i in 1:vars) vec.plot(rfo2,X,i,zoom=1.5,limitY=TRUE)

#plotvariable X3 and X4 with vec.plot
Col = fcol(X,3:4)
vec.plot(rfo2,X,3:4,zoom=1,grid.lines=100,col=Col)

## End(Not run)
```

---

`Xtestmerger` *merge training set (X) and (test) set*

---

## Description

... and expand inbag matrix and training target vector to compute FC for a test set.

## Usage

```
Xtestmerger(X,test,inbag=NULL,y=NULL)
```

## Arguments

| | |
|---|---|
| `X` | X , training set data.frame used to train a random forest model |
| `test` | test, a test set data.frame which feature contributions should be computed for |
| `inbag` | matrix of inbag sampling to expande with training set, which is set OOB for any tree |
| `y` | random forest target vector, which is set to first value for observation |

## Details

Xtestmerger is a low-level function to merge a test set with X training set. There can be no names, column class, column number mismatch. Moreover any level in any factor of test must be present in X, as RF/forestFloor cannot score a unknown factor level / category.

## Value

List of merged bigX, bigInbag and bigy. The two latter may be NULL if not provided.

## Author(s)

Soren Havelund Welling

## Examples

```
library(randomForest)
library(forestFloor)
#X y could be a training set
X = data.frame(numeric = c(1,5,2,7,-4.3),
               factor1 = factor(c("jim","freddy","marley","marley","alfred")),
               factor2 = factor(c("jill","ann","liz","leila","vicky")))
y = factor(1:5)
set.seed(1)
rf = randomForest(X,y,keep.inbag=TRUE,ntree=7)
#should not raise any error
test = data.frame(numeric = rnorm(5),
                  factor1 = factor(c("jim","jim","jim","freddy","freddy")),
                  factor2 = factor(c("jill","jill","vicky","leila","vicky"))
                  )
out = Xtestmerger(X,test,inbag=rf$inbag,y=y)
```

# Index